Fail2ban Evaluierung

Paul Raatschen

1. Dezember 2022

Übersicht

- Grundlagen
  - Fail2ban
  - eBPF
- Messungen
  - Raw UDP Traffic
  - DNS Traffic
  - HTTP Traffic

Fail2ban

- Open Source Intrusion Prevention System.
- Blockiert ungewollten Datenverkehr auf Basis von Anwendungs-Logdateien.
- Fail2ban erlaubt die Konfiguration von “Jails”.
- Ein Jail besteht aus Filter, Action sowie Paramtern.

Jail Parameter und Filter

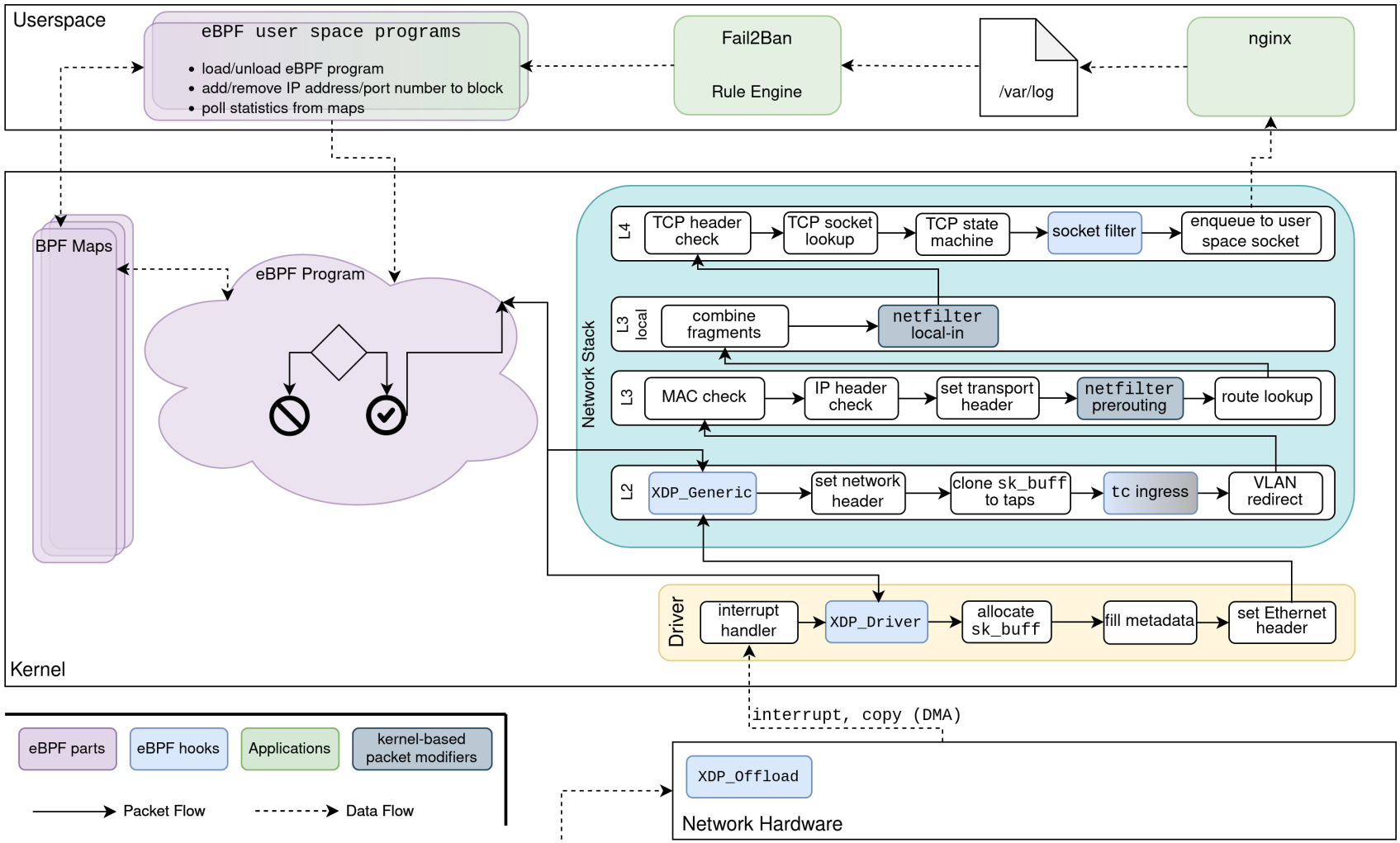
```
# Jail Definition
[udp-testsvr]
port    = 8080
logpath = /mnt/scratch/PR/udpsvr.log
enabled = true
filter  = udp-testsvr
findtime = 10
bantime = 180
action = xdp
maxretry = 0

# Filter
[Definition]
failregex = Address = <HOST>, Port = \d{1,5}, Payload = 2\d{2}
datepattern =  %%Y-%%b-%%d %%H:%%M:%%S
```

eBPF

- Extended Berkley Packet Filter. Interface im Linux Kernel.
- Erlaubt ereignisbasierte Ausführung von Benutzerprogrammen im Kernel, u.a. für Paketfilterung.
- Kann für Fail2ban an Stelle von Iptables verwendet werden.

Fail2ban + eBPF



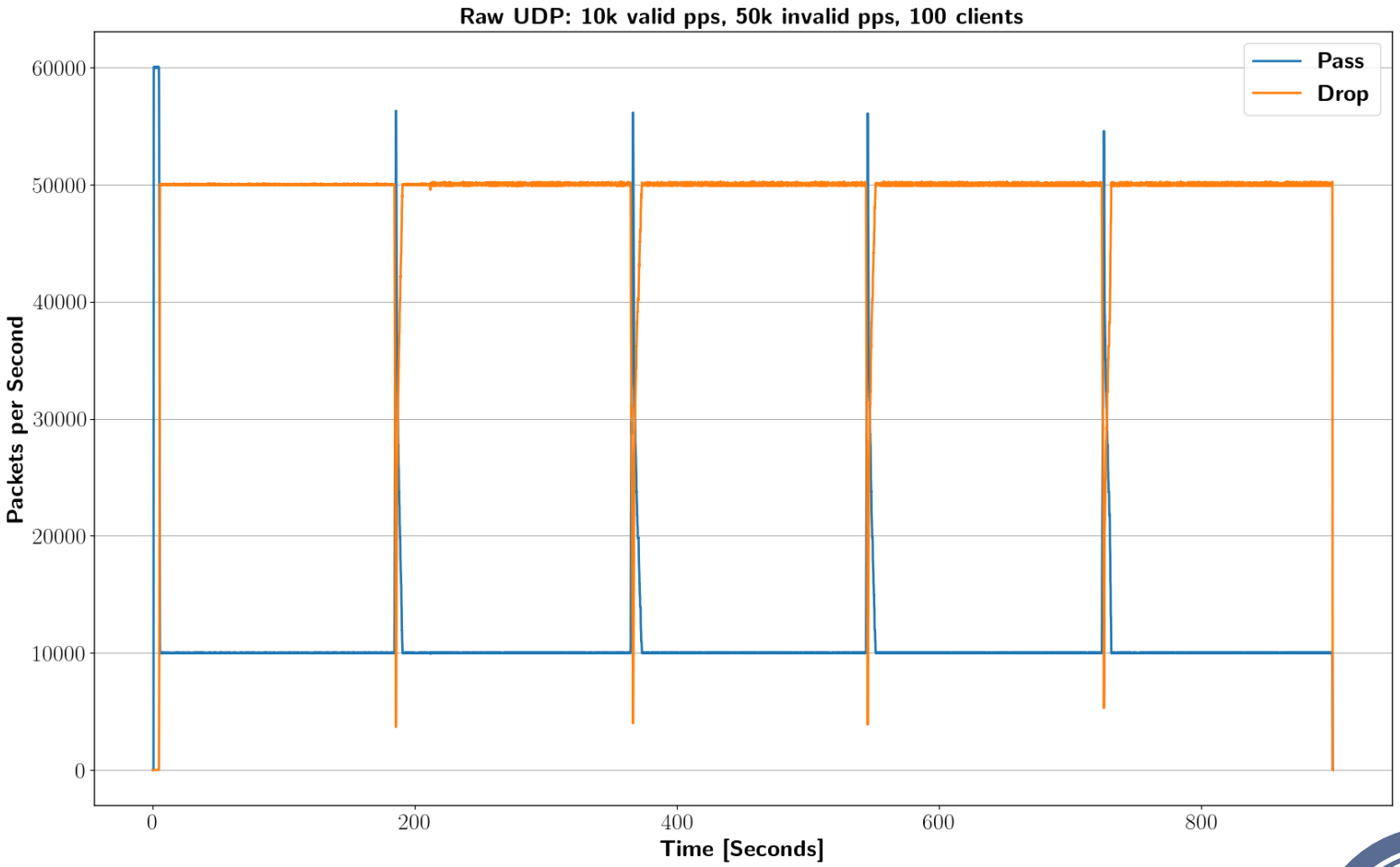
Quelle: Florian Mikolajczak : “Implementation and Evaluation of an Intrusion Prevention System Leveraging eBPF on the Basis of Fail2Ban”, Master Thesis, 2022

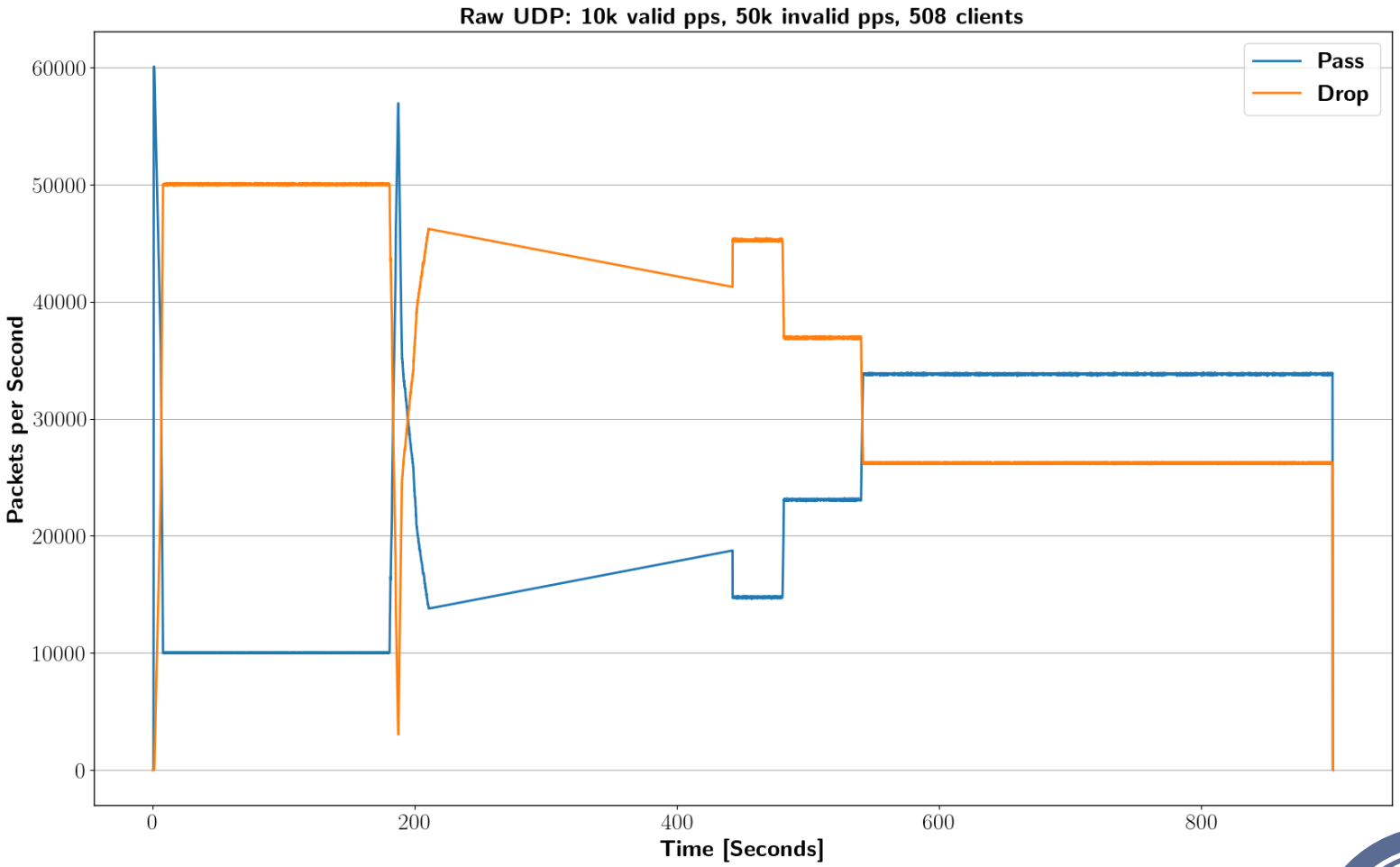
Testumgebung

Hardware	
CPU	16 x Intel Xeon Silver 4314 CPU @ 2.4GHz
NIC	Mellanox ConnectX-5 100Gb/s Ethernet
RAM	131.4GB
Software	
OS	Debian 10
Kernel	5.15.7
NIC Driver	mlx5_core 5.5-1.0.3
BIND	9.11.5
Nginx	1.14.2
Fail2Ban	0.11.2
TRex	2.99

Raw UDP Traffic

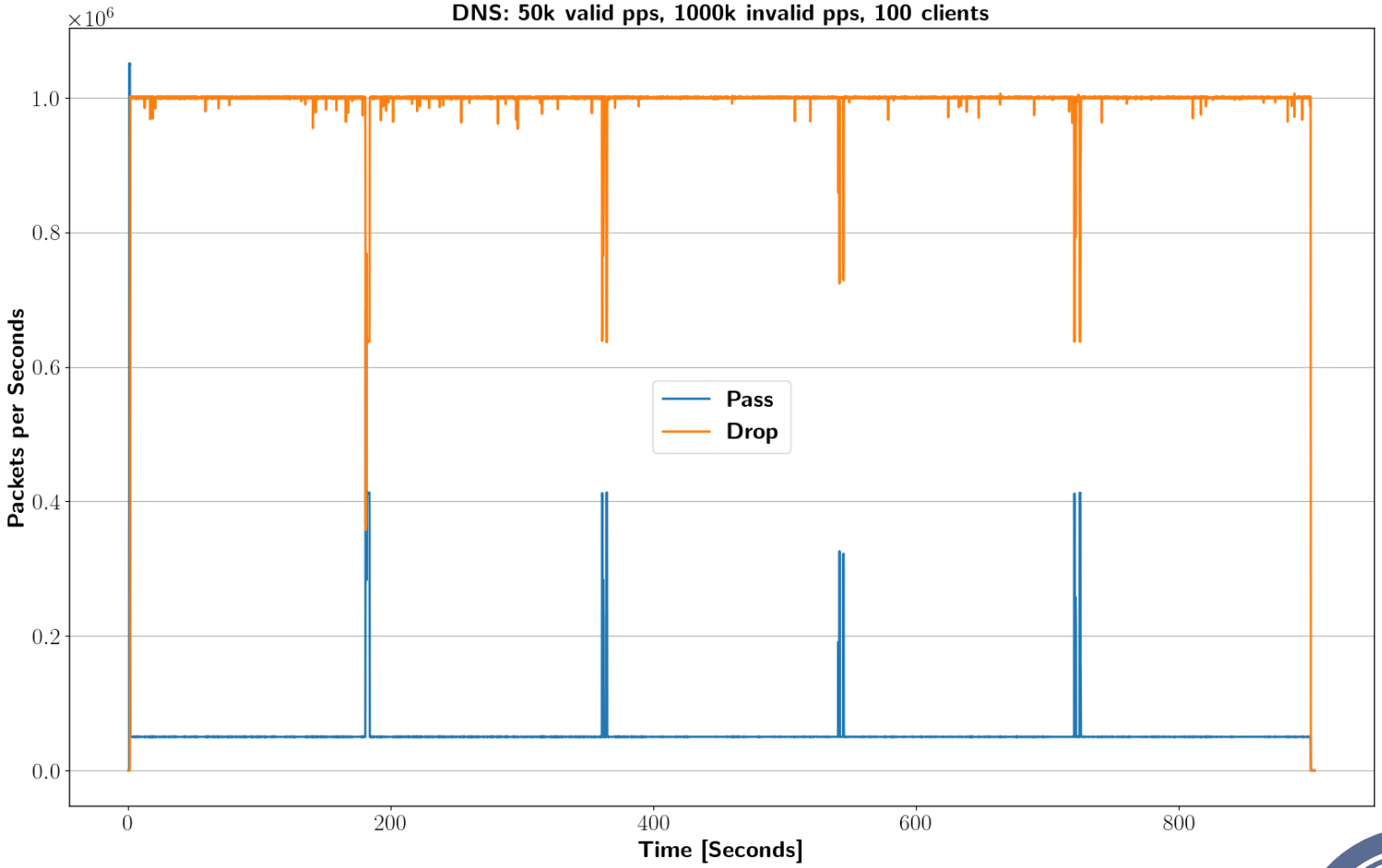
- Minimaler (single-threaded) Server, der UDP Anfragen beantwortet.
- Problem: Schwache Performance, schafft nur $\sim$100k Anfragen pro Sekunde.
- Messung: Client (Trex auf bsnode2) sendet UDP Pakete mit 1 Byte Payload an Server (bsnode1).
- Mischung aus validem und invalidem Traffic.
- Variablen: Packets per second (PPS) und Anzahl der Clients.

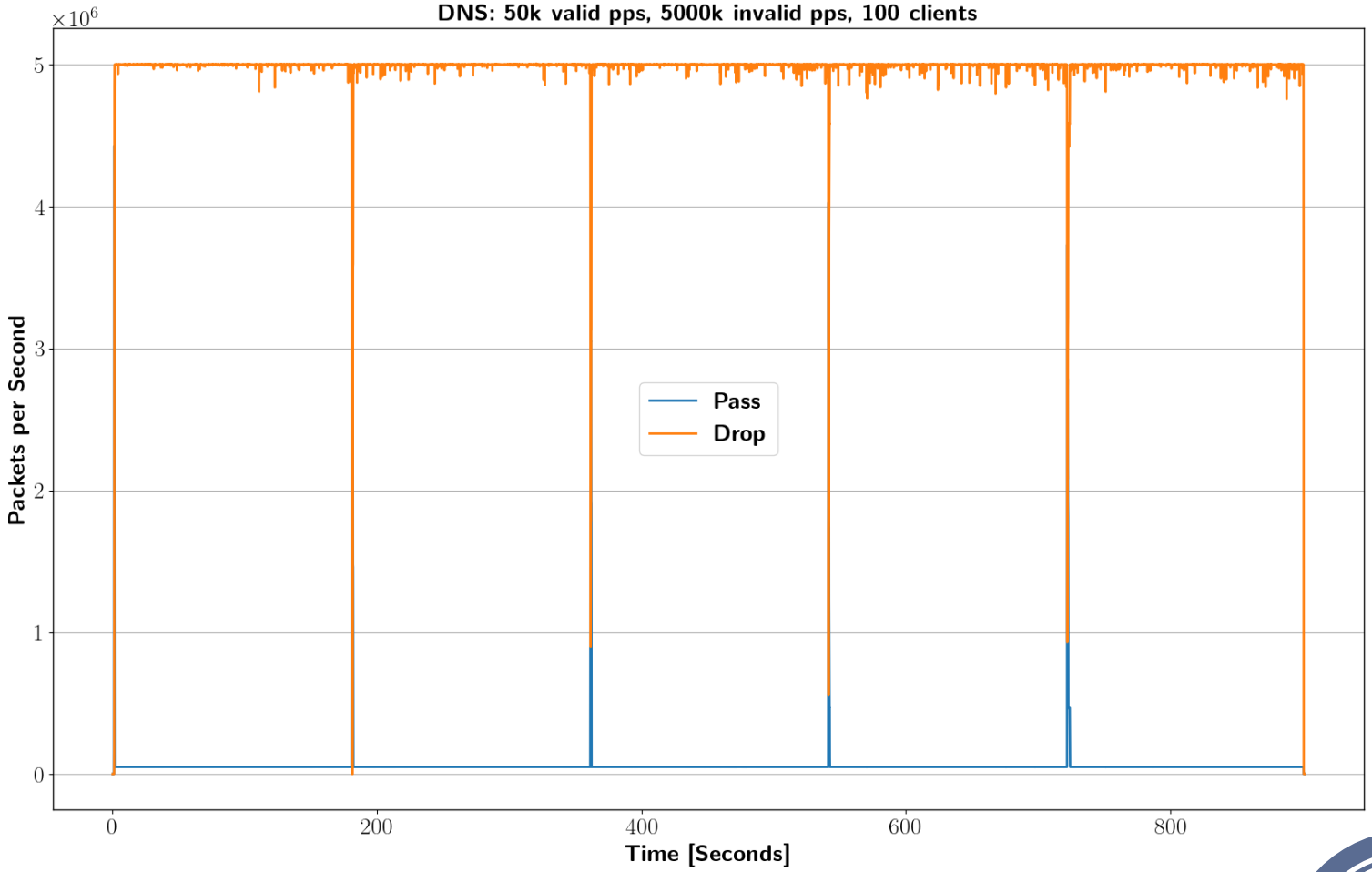

Raw UDP: 10k valid pps, 50k invalid pps, 100 clients
Pass
Drop
Packets per Second
60000
50000
40000
30000
20000
10000
0
0
200
400
600
800
Time [Seconds]

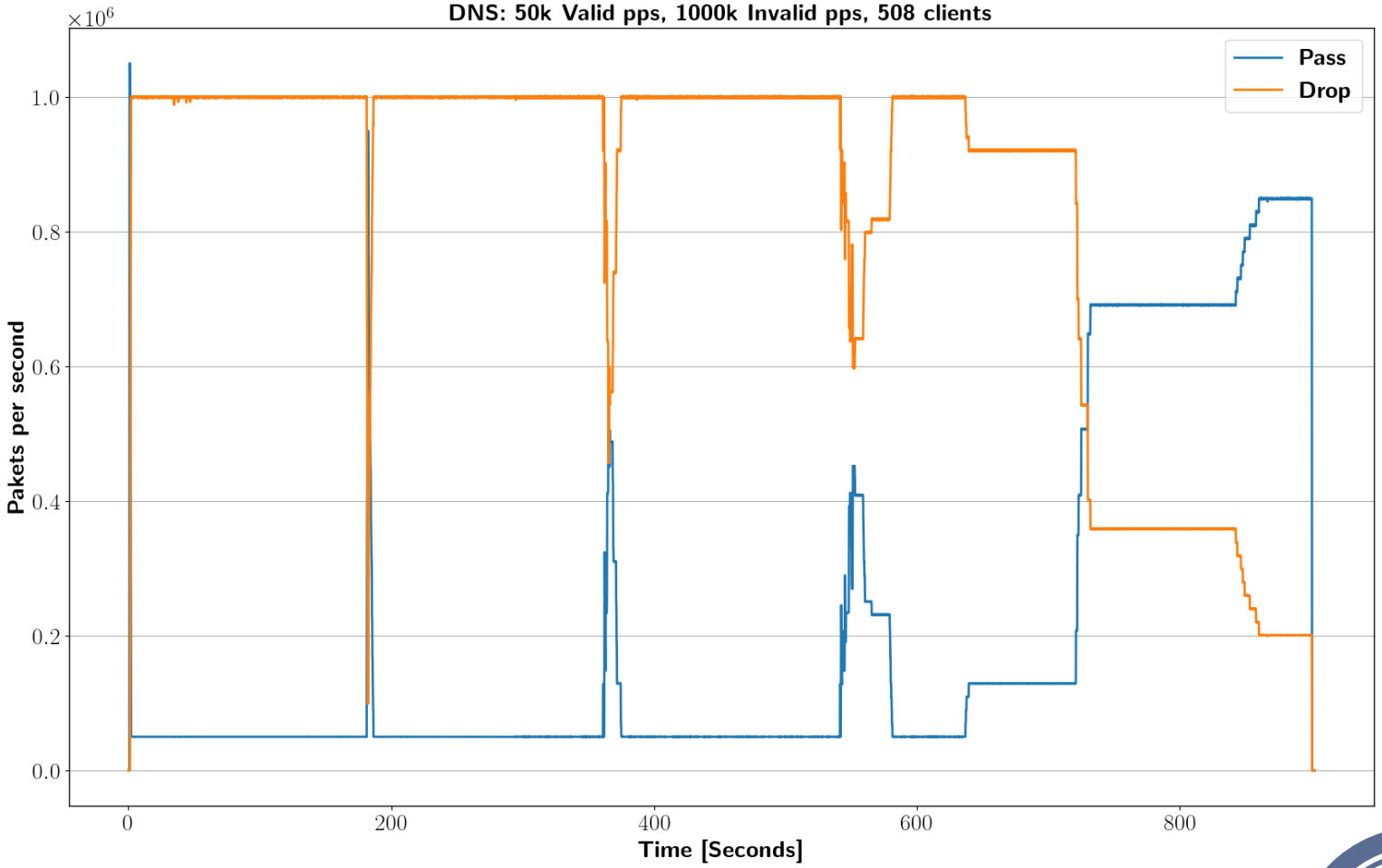

Raw UDP: 10k valid pps, 50k invalid pps, 508 clients
Pass
Drop
Packets per Second
60000
50000
40000
30000
20000
10000
0
0
200
400
600
800
Time [Seconds]

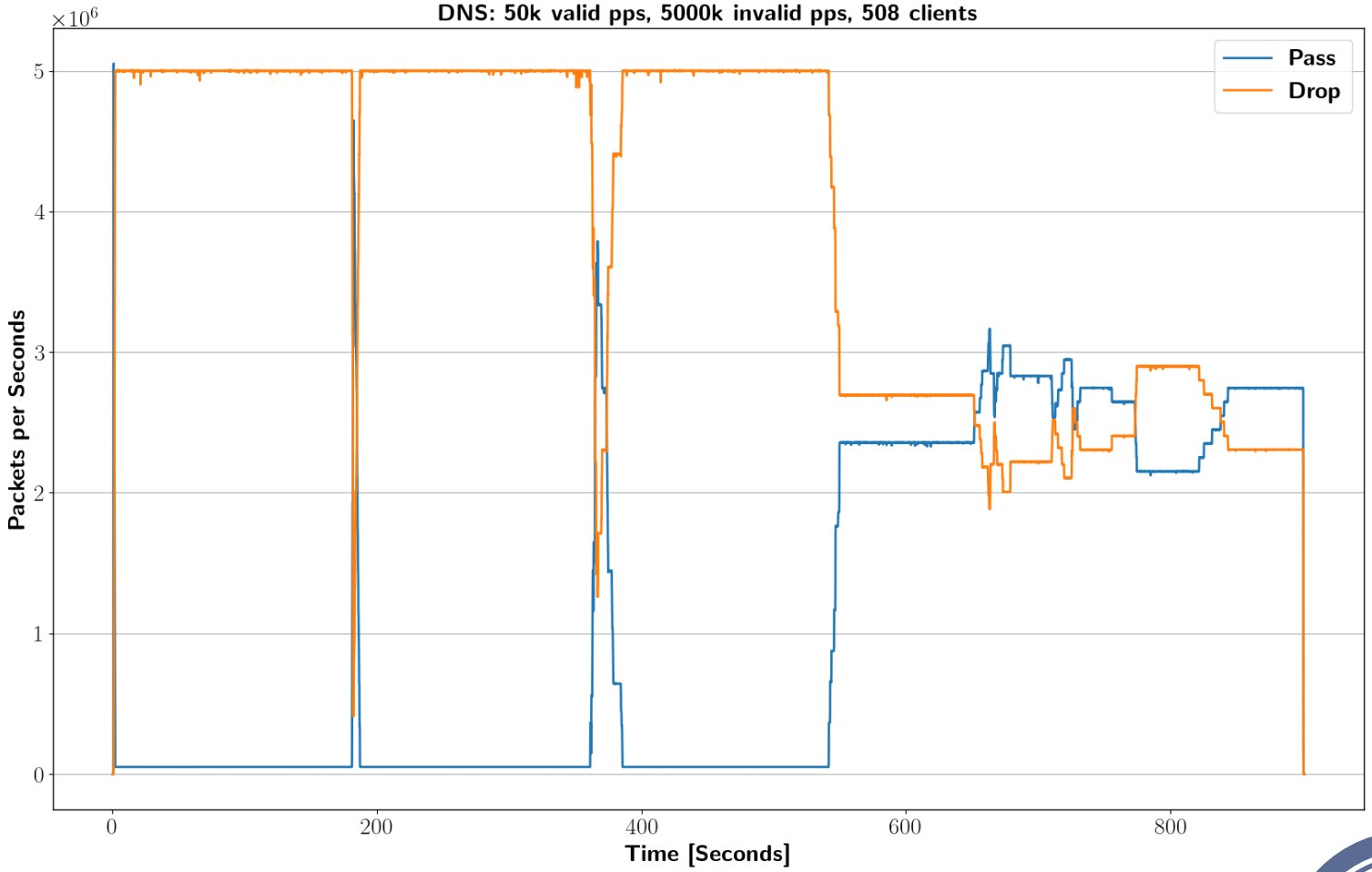
DNS Traffic

- Client (Trex auf bsnode2) schickt DNS Requests an BIND Server (bsnode1).
- Variablen: Packets per Second (PPS) und Anzahl der Clients.


DNS: 50k valid pps, 1000k invalid pps, 100 clients
×10^6
1.0
0.8
0.6
0.4
0.2
0.0
Packets per Seconds
Pass
Drop
0
200
400
600
800
Time [Seconds]

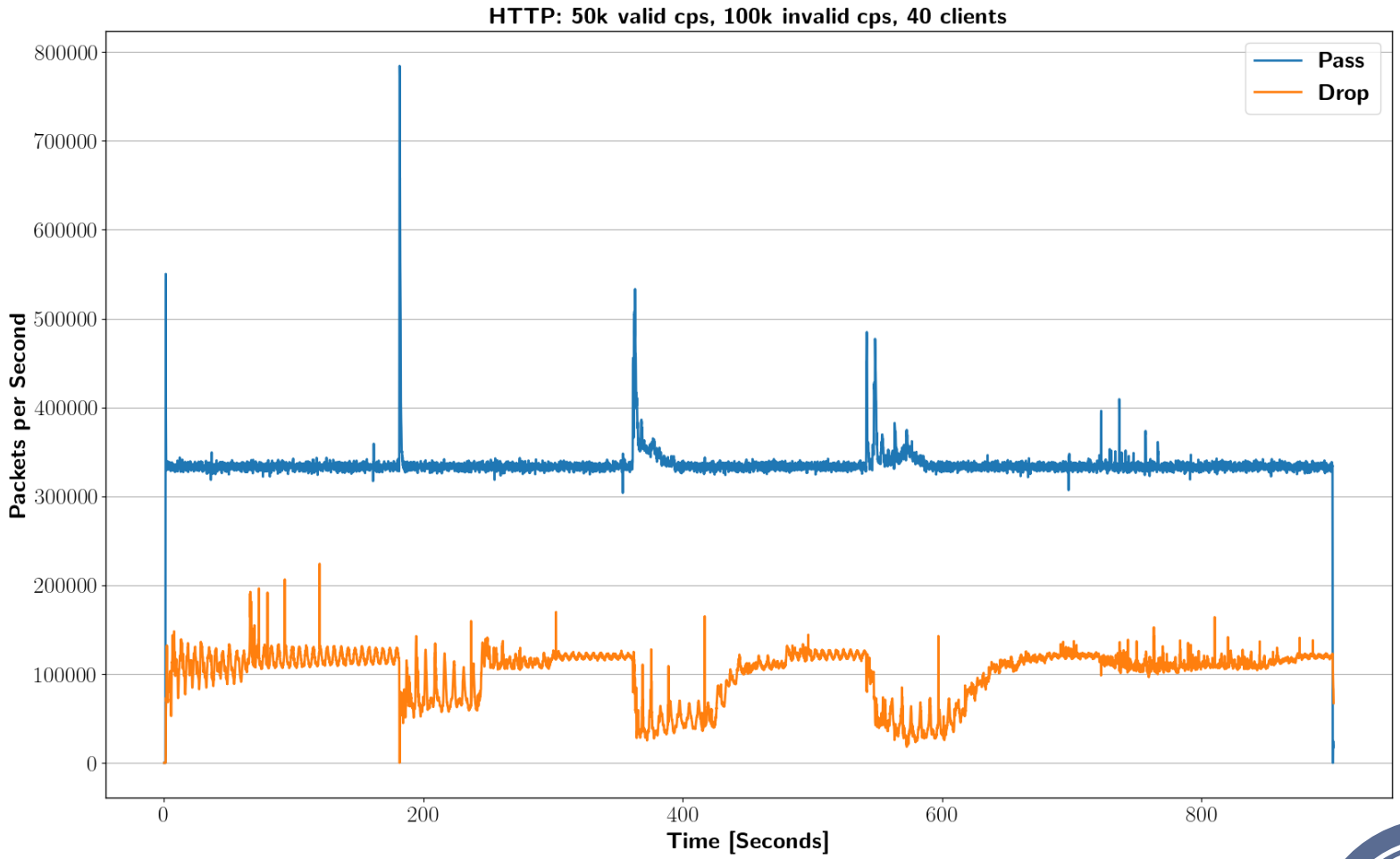

DNS: 50k valid pps, 5000k invalid pps, 100 clients
×10^6
Packets per Second
0
1
2
3
4
5
0
200
400
600
800
Time [Seconds]
Pass
Drop

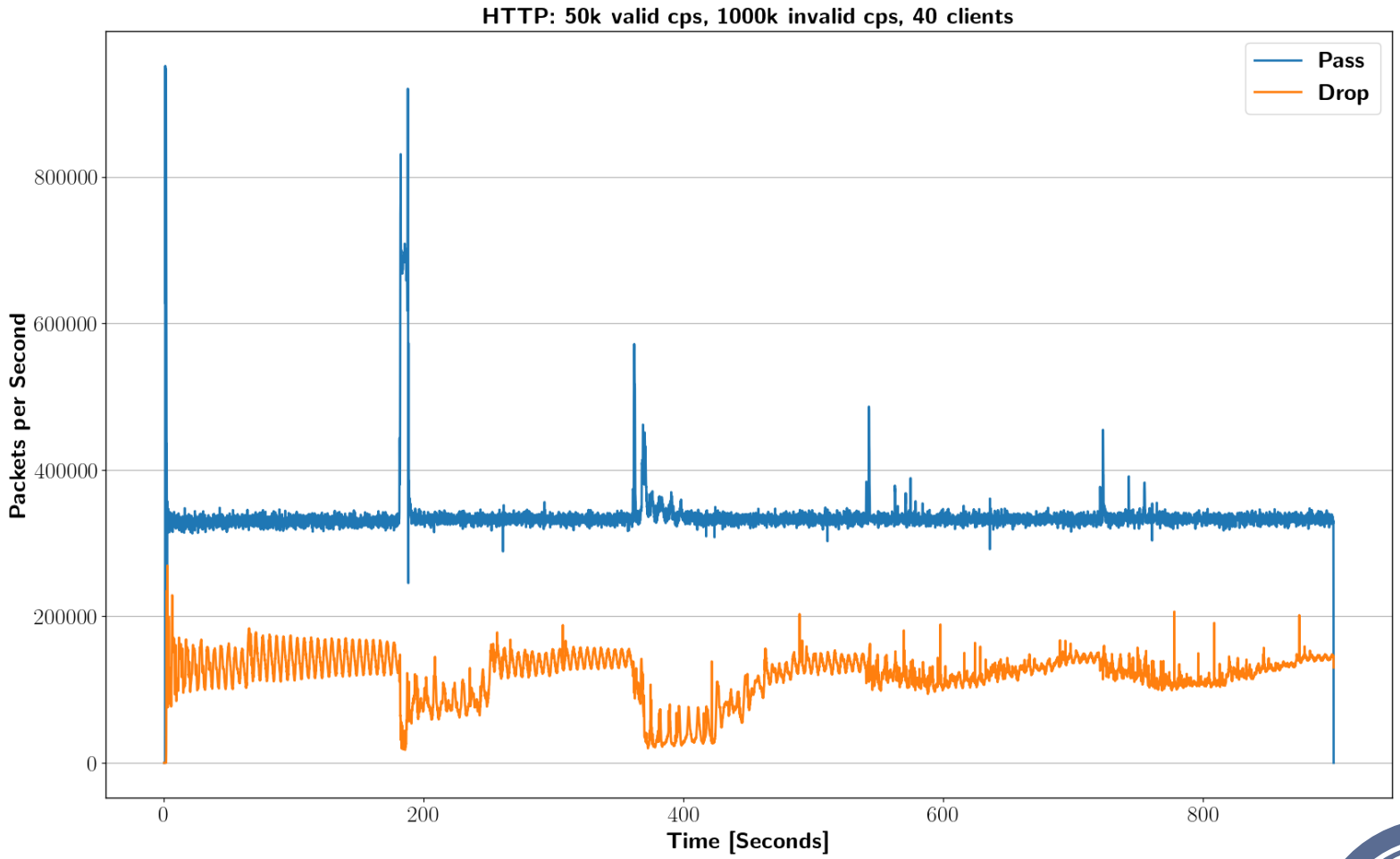

DNS: 50k Valid pps, 1000k Invalid pps, 508 clients
×10^6
1.0
0.8
0.6
0.4
0.2
0.0
Pakets per second
0
200
400
600
800
Time [Seconds]
Pass
Drop


DNS: 50k valid pps, 5000k invalid pps, 508 clients
×10^6
Packets per Seconds
Time [Seconds]
Pass
Drop
0
1
2
3
4
5
0
200
400
600
800

HTTP Traffic

- Client (Trex auf bsnode2) schickt HTTP GET Request an Nginx Server (bsnode1).
- 8 TCP Pakete pro Verbindung (inklusive Handshake).
- Variablen: Connections per second (CPS).
- Problem: Trex liefert nicht die spezifizierte Anzahl an CPS.


HTTP: 50k valid cps, 100k invalid cps, 40 clients
Pass
Drop
Packets per Second
800000
700000
600000
500000
400000
300000
200000
100000
0
0
200
400
600
800
Time [Seconds]


HTTP: 50k valid cps, 1000k invalid cps, 40 clients
Pass
Drop
Packets per Second
800000
600000
400000
200000
0
0
200
400
600
800
Time [Seconds]